

作业提交服务器: ftp://192.168.134.123

用户名: uploader

密码: sdt7%5252@3

! 注意事项

1. 每题需要提交一个cpp文件, 确保cpp文件可以正确编译。cpp文件用题目编号命名, 除非有特别说明。
2. 每题需要提交至少一张运行结果截图。将所有题目的截图放在一个pdf文件中。
3. 将所有cpp文件和一个pdf文件打包成zip文件, 用自己的学号命名, 并上传至ftp指定文件夹。

举例说明: 比如学号是1001, 本次有3个题目, 那么最终提交1个1001.zip文件, 其中包含3个cpp文件(分别是1.cpp, 2.cpp, 3.cpp)和1个pdf文件, 其中pdf文件中包含至少3个运行结果的截图。第1次作业上传至hm1文件夹。

第4次作业

提交时间: 2024年6月2日22:00

1. 在程序的执行过程中, 构造函数与析构函数都将在明确的、可预计的时间点上被调用。我们可以更简单的看待构造函数和析构函数的调用: 每当类型X的一个对象被构建时, 类型X的一个构造函数将被调用; 每当类型X的一个对象被销毁时, 类型X的析构函数将被调用。

每当一个类对象被销毁时, 该类的析构函数将被调用: 这种情况可能发生在变量的作用域结束时、程序结束时或者delete一个指向对象的指针时。

每当一个类对象被构建时, 该类的某个构造函数将被调用: 这种情况可能发生在变量初始化时、用new创建对象或者拷贝对象时。

理解它们的一个好办法是向构造函数、赋值操作和析构函数添加打印语句, 然后尝试运行程序, 如下所示

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  struct X {
6      int val;
7      void out(const std::string& s, int nv)
8      {
9          std::cerr << this << "->" << s << ": " << val << "(" << nv << ")\n";
10     }
11
12     X() { out("X()", 0); val = 0; }
13     X(int v) { val = v; out("X(int)", v); }
14     X(const X& x) { val = x.val; out("X(X&)", x.val); }
15     X& operator=(const X& a)
16     {
17         out("X::operator=()", a.val);
18         val = a.val;
```

```

19         return *this;
20     }
21     ~X() { out("~X()", 0); }
22 };
23
24 X glob(2);
25
26 X copy(X a) { return a; }
27
28 X copy2(X a) { X aa = a; return aa; }
29
30 X& ref_to(X& a) { return a; }
31
32 X* make(int i) { X a(i); return new X(a); }
33
34 struct XX { X a; X b; };
35
36 void print(int i)
37 {
38     std::cout << "Line " << i << ": ";
39     for (int i = 0; i < 20; ++i) std::cout << "-";
40     std::cout << '\n';
41 }
42
43 int main()
44 {
45     print(0);
46     X loc { 4 };
47     print(1);
48     X loc2 { loc };
49     print(2);
50     loc = X { 5 };
51     print(3);
52     loc2 = copy(loc);
53     print(4);
54     loc2 = copy2(loc);
55     print(5);
56     X loc3 { 6 };
57     print(6);
58     X& r = ref_to(loc);
59     print(7);
60     delete make(7);
61     print(8);
62     delete make(8);
63     print(9);
64     std::vector<X> v(4);
65     print(10);
66     XX loc4;
67     print(11);
68     X* p = new X { 9 };
69     print(12);
70     delete p;

```

```

71     print(13);
72     X* pp = new X[5];
73     print(14);
74     delete[] pp;
75     print(15);
76     return 0;
77 }

```

在你的机器上运行上述程序，并确保你能看懂输出结果的含义。本题仅需提交一个pdf文件，说明你使用了什么编译器、运行结果的截图以及对输出结果你感到疑惑的地方。

2. 完善我们自己的Vector类。头文件vector.hpp如下所示，请在源文件vector.cpp中完成所有函数的定义（注意：swap函数的定义放在头文件中），并在use.cpp文件中使用Vector类。

```

1  //vector.hpp
2
3  class Vector {
4      friend void swap(Vector& a, Vector& b);
5  private:
6      size_t sz; //元素数量
7      double* elem; //指向第一个元素的指针
8      size_t space; //已分配内存能存放元素的数量
9  public:
10     Vector();
11     explicit Vector(size_t s); //构造函数
12     Vector(size_t s, double v); //构造函数
13     Vector(std::initializer_list<double> lst); //构造函数
14     Vector(const Vector& a); //拷贝构造函数
15     Vector(Vector&& a); //移动构造函数
16     Vector& operator=(const Vector& a); //拷贝赋值
17     Vector& operator=(Vector&& a); //移动赋值
18     ~Vector(); //析构函数
19     size_t size() const;
20     double& operator[](size_t n);
21     const double& operator[](size_t n) const;
22     void reserve(size_t new_space);
23     void resize(size_t new_size);
24     size_t capacity() const;
25     void push_back(double d);
26 };
27
28 std::ostream& operator<<(std::ostream& os, const Vector& v);
29
30 inline void swap(Vector& a, Vector& b)
31 {
32     //add your code here
33 }

```

```

1  //use.cpp
2  #include "Vector.hpp"
3  #include <iostream>

```

```

4
5 Vector f(size_t n)
6 {
7     Vector v(n);
8     for (size_t i { 0 }; i < v.size(); ++i)
9         v[i] = 1.1 * i;
10    return v;
11 }
12
13 int main()
14 {
15     Vector v;
16     for (int i { 0 }; i < 20; ++i) {
17         std::cout << "size = " << v.size() << ", capacity = " << v.capacity() <<
std::endl;
18         v.push_back(i);
19     }
20     Vector u { 10, 20, 30, 40 };
21     for (size_t i { 0 }; i < u.size(); ++i)
22         std::cout << u[i] << ' ';
23     std::cout << std::endl;
24
25     std::cout << "size(u) = " << u.size() << ", capacity(u) = " << u.capacity()
<< std::endl;
26     u = v;
27     std::cout << "size(u) = " << u.size() << ", capacity(u) = " << u.capacity()
<< std::endl;
28
29     std::cout << u << std::endl;
30     u.resize(5);
31     std::cout << u << std::endl;
32
33     u = f(8);
34     std::cout << u << std::endl;
35
36     Vector* p { new Vector(3, 3.14) };
37     std::cout << *p << std::endl;
38
39     swap(*p, u);
40     std::cout << u << std::endl;
41     std::cout << *p << std::endl;
42
43     delete p;
44     p = nullptr;
45 }

```